Theoretische Informatik

Beweise 101

Nicolas Wehrli, ETH Zurich

October 20, 2023

Contents

1 Grundbegriffe

Für eine Menge A bezeichnet $|A|$ die Kardinalität von A und $\mathcal{P}(A) = \{S \mid S \subseteq A\}$ die Potenzmenge von A.

In diesem Kurs definieren wir $\mathbb{N} = \{0, 1, 2, \dots\}$.

1.1 Alphabet

Definition Alphabet

Eine endliche, nichtleere Menge Σ heisst **Alphabet**. Die Elemente eines Alphabets werden **Buchstaben (Zeichen, Symbole)** genannt.

Beispiele

- $\Sigma_{\text{bool}} = \{0, 1\}$
- $\Sigma_{\text{lat}} = \{a, ..., z\}$
- $\Sigma_{\text{Tastatur}} = \Sigma_{\text{lat}} \cup \{A, ..., Z, \text{␣}, >, <, (,), ..., !\}$
- $\Sigma_{\text{logic}} = \{0, 1, (,), \wedge, \vee, \neg\}$
- $\Sigma_{abc} = \{a, b, c\}$ (**unser Beispiel für weitere Definitionen**)

1.2 Wort

Definition Wort

- Sei Σ ein Alphabet. Ein **Wort** über Σ ist eine endliche (eventuell leere) Folge von Buchstaben aus Σ.
- Das **leere Wort** λ ist die leere Buchstabenfolge.
- Die **Länge** $|w|$ eines Wortes w ist die Länge des Wortes als Folge, i.e. die Anzahl der Vorkommen von Buchstaben in w.
- Σ^* ist die Menge aller Wörter über Σ. $\Sigma^+ := \Sigma^* \setminus \{\lambda\}$ ist Menge aller nichtleeren Wörter über Σ.
- Seien $x \in \Sigma^*$ und $a \in \Sigma$. Dann ist $|x|_a$ definiert als die Anzahl der Vorkommen von a in x.

Achtung Metavariablen! I.e. Das a in der Definition ist steht für einen beliebigen Buchstaben aus Σ und **nicht** nur für den Buchstaben 'a', der in Σ sein könnte.

Bemerkungen

- Wir schreiben Wörter ohne Komma, i.e. eine Folge $x_1, x_2, ..., x_n$ schreiben wir $x_1x_2...x_n$.
- $|\lambda| = 0$ aber $|\text{␣}| = 1$ von Σ_{Tastatur}.
- Der Begriff **Wort** als Fachbegriff der Informatik entspricht **nicht** der Bedeutung des Begriffs Wort in natürlichen Sprachen!
- E.g. Mit ␣ kann der Inhalt eines Buches oder ein Programm als ein Wort über Σ_{Tastatur} betrachtet werden.

Beispiel Verschiedene Wörter über Σ_{abc}:

a, aa, aba, cba, $caaaab$ etc.

Die **Verkettung (Konkatenation)** für ein Alphabet Σ ist eine Abbildung Kon: $\Sigma^* \times \Sigma^* \to \Sigma^*$, so dass

$$\text{Kon}(x, y) = x \cdot y = xy$$

für alle $x, y \in \Sigma^*$.

- Die Verkettung Kon (i.e. Kon von einem Kon (über das gleiche Alphabet Σ)) ist eine assoziative Operation über Σ^*.

$$\text{Kon}(u, \text{Kon}(v, w)) = \text{Kon}(\text{Kon}(u, v), w), \ \forall u, v, w \in \Sigma^*$$

- $x \cdot \lambda = \lambda \cdot x = x, \ \forall x \in \Sigma^*$
- $\implies$ (Σ^*, Kon) ist ein Monoid mit neutralem Element λ.
- Kon nur kommutativ, falls $|\Sigma| = 1$.
- $|xy| = |x \cdot y| = |x| + |y|$. (Wir schreiben ab jetzt xy statt Kon(x, y))

Beispiel

Wir betrachten wieder Σ_{abc}. Sei $x = abba$, $y = cbcbc$, $z = aaac$.

- $\text{Kon}(x, \text{Kon}(y, z)) = \text{Kon}(x, yz) = xyz = abbacbcbcaaac$
- $|xy| = |abbacbcbc| = 9 = 4 + 5 = |abba| + |cbcbc| = |x| + |y|$

Für eine Wort $a = a_1 a_2 ... a_n$, wobei $\forall i \in \{1, 2, ..., n\}$. $a_i \in \Sigma$, bezeichnet $a^{\text{R}} = a_n a_{n-1} ... a_1$ die **Umkehrung (Reversal)** von a.

Sei Σ ein Alphabet. Für alle $x \in \Sigma^*$ und alle $i \in \mathbb{N}$ definieren wir die i-te **Iteration** x^i von x als

$$x^0 = \lambda, x^1 = x \text{ und } x^i = xx^{i-1}.$$

Beispiel

Wir betrachten wieder Σ_{abc}. Sei $x = abba$, $y = cbcbc$, $z = aaac$.

- $z^{\text{R}} = (aaac)^{\text{R}} = caaa$
- $x^{\text{R}} = (abba)^{\text{R}} = abba$
- $x^0 = \lambda$
- $y^2 = yy^{2-1} = yy = cbcbccbcbc$
- $z^3 = zz^2 = zzz = aaacaaacaaac$
- $(x^{\text{R}} z^{\text{R}})^{\text{R}} = ((abba)^{\text{R}}(aaac)^{\text{R}})^{\text{R}} = (abbacaaa)^{\text{R}} = aaacabba$

Seien $v, w \in \Sigma^*$ für ein Alphabet Σ.

- v heisst ein **Teilwort** von $w \iff \exists x, y \in \Sigma^* : \ w = xvy$
- v heisst ein **Präfix** von $w \iff \exists y \in \Sigma^* : \ w = vy$
- v heisst ein **Suffix** von $w \iff \exists x \in \Sigma^* : \ w = xv$
- $v \neq \lambda$ heisst ein **echtes** Teilwort (Präfix, Suffix) von $w \iff v \neq w$ und v Teilwort(Präfix, Suffix) von w

Beispiel

Wir betrachten wieder Σ_{abc}. Sei $x = abba$, $y = cbcbc$, $z = aaac$.

- bc ist ein echtes Suffix von y
- $abba$ ist kein echtes Teilwort von x.
- $cbcb$ ist ein echtes Teilwort und echtes Präfix von y.
- ac ist ein echtes Suffix.
- $abba$ ist ein Suffix, Präfix und Teilwort von x.

Aufgabe 1

Sei Σ ein Alphabet und sei $w \in \Sigma^*$ ein Wort der Länge $n \in \mathbb{N} \setminus \{0\}$. Wie viele unterschiedliche Teilwörter kann w höchstens haben?

Lösung

Wir haben $w = w_1w_2...w_n$ mit $w_i \in \Sigma$ für $i = 1, ..., n$. Wie viele Teilwörter beginnen mit w_1? Wie viele Teilwörter beginnen mit w_2?

Wir haben also $n + (n-1) + ... + 1 = \frac{n(n+1)}{2}$ Teilwörter. Etwas fehlt aber in unserer Berechnung...

Das leere Wort λ ist auch ein Teilwort! Also haben wir $\frac{n(n+1)}{2} + 1$ Teilwörter.

Aufgabe 2

Sei $\Sigma = \{a, b, c\}$ und $n \in \mathbb{N}$. Bestimme die Anzahl der Wörter aus Σ^n, die das Teilwort a enthalten.

Lösung

In solchen Aufgaben ist es manchmal einfach, das Gegenteil zu berechnen und so auf die Lösung zu kommen. Wie viele Wörter aus Σ^n enthalten das Teilwort a **nicht**?

Da wir jetzt die Anzahl Wörter der Länge n wollen, die nur b und c enthalten, kommen wir auf $|\{b, c\}|^n = 2^n$.

Daraus folgt, dass genau $|\Sigma|^n - 2^n = 3^n - 2^n$ Wörter das Teilwort a enthalten.

Aufgabe 3

Sei $\Sigma = \{a, b, c\}$ und $n \in \mathbb{N} \setminus \{0\}$. Bestimme die Anzahl der Wörter aus Σ^n, die das Teilwort aa nicht enthalten.

Lösung

Wir bezeichnen die Menge aller Wörter mit Länge n über Σ, die aa nicht enthalten als L_n.

Schauen wir mal die ersten zwei Fälle an:

- $L_1 = \{a, b, c\} \implies |L_1| = 3$
- $L_2 = \{ab, ac, ba, bb, bc, ca, cb, cc\} \implies |L_2| = 8$

Nun können wir für $m \geq 3$ jedes Wort $w \in L_m$ als Konkatination $w = x \cdot y \cdot z$ schreiben, wobei wir zwei Fälle unterscheiden:

(a) $\boldsymbol{z \neq a}$

In diesem Fall kann $y \in \{a, b, c\}$ sein, ohne dass die Teilfolge aa entsteht und somit ist xy ein beliebiges Wort aus L_{m-1}.

Dann könnten wir alle Wörter in diesem Case durch $L_{m-1} \cdot \{b, c\}$ beschreiben, was uns die Kardinalität $2 \cdot |L_{m-1}|$ gibt.

(b) $\boldsymbol{z = a}$

In diesem Fall muss $y \neq a$ sein, da sonst aa entstehen würde.

Somit kann xy nur in b oder c enden. x kann aber ein beliebiges Wort der Länge $m-2$ sein.

Deshalb können wir alle Wörter in diesem Case durch $L_{m-2} \cdot \{b, c\} \cdot \{a\}$ beschreiben. Kardinalität: $2 \cdot |L_{m-2}|$.

Daraus folgt

$$|L_n| = \begin{cases} 3 & n = 1 \\ 8 & n = 2 \\ 2|L_{n-1}| + 2|L_{n-2}| & n \geq 3 \end{cases}$$

Sei $\Sigma = \{s_1, s_2, ..., s_m\}, m \geq 1$, ein Alphabet und sei $s_1 < s_2 < ... < s_m$ eine Ordnung auf Σ. Wir definieren die **kanonische Ordnung** auf Σ^* für $u, v \in \Sigma^*$ wie folgt:

$$u < v \iff |u| < |v| \vee (|u| = |v| \wedge u = x \cdot s_i \cdot u' \wedge x \cdot s_j \cdot v')$$
$$\text{für irgendwelche } x, u', v' \in \Sigma^* \text{ und } i < j.$$

Sei $\Sigma_{abc} = \{a, b, c\}$ und wir betrachten folgende Ordnung auf Σ_{abc}: $c < a < b$.

Was wäre die kanonische Ordnung folgender Wörter?

c, abc, $aaac$, $aaab$, $bacc$, a, λ

λ, c, a, abc, $aaac$, $aaab$, $bacc$

1.3 Sprache

Eine **Sprache** L über einem Alphabet Σ ist eine Teilmenge von Σ^*.

- Das Komplement $L^{\complement}$ der Sprache L bezüglich Σ ist die Sprache $\Sigma^* \setminus L$.
- $L_\emptyset = \emptyset$ ist die **leere Sprache**.
- $L_\lambda = \{\lambda\}$ ist die einelementige Sprache, die nur aus dem leeren Wort besteht.

Konkatenation von Sprachen

Sind L_1 und L_2 Sprachen über Σ, so ist

$$L_1 \cdot L_2 = L_1 L_2 = \{vw \mid v \in L_1 \text{ und } w \in L_2\}$$

die **Konkatenation** von L_1 und L_2.

Ist L eine Sprache über Σ, so definieren wir

$$L^0 := L_\lambda \text{ und } L^{i+1} := L^i \cdot L \text{ für alle } i \in \mathbb{N},$$
$$L^* = \bigcup_{i \in \mathbb{N}} L^i \text{ und } L^+ = \bigcup_{i \in \mathbb{N}\setminus\{0\}} L^i = L \cdot L^*.$$

L^* nennt man den **Kleene'schen Stern** von L.

Man bemerke, dass $\Sigma^i = \{x \in \Sigma^* \mid |x| = i\}$, $L_\emptyset L = L_\emptyset = \emptyset$ und $L_\lambda \cdot L = L$.

Mögliche Sprachen über Σ_{abc}

- $L_1 = \emptyset$
- $L_2 = \{\lambda\}$
- $L_3 = \{\lambda, ab, baca\}$
- $L_4 = \Sigma^*_{abc}$, $L_5 = \Sigma^+_{abc}$, $L_6 = \Sigma_{abc}$ oder $L_7 = \Sigma^{27}_{abc}$
- $L_8 = \{c\}^* = \{c^i \mid i \in \mathbb{N}\}$
- $L_9 = \{a^p \mid p \text{ ist prim.}\}$
- $L_{10} = \{c^i a^{3i^2} b a^i c \mid i \in \mathbb{N}\}$

λ ist ein Wort über jedes Alphabet. Aber es muss nicht in jeder Sprache enthalten sein!

Seien L_1, L_2 und L_3 Sprachen über einem Alphabet Σ. Dann gilt

$$L_1L_2 \cup L_1L_3 = L_1(L_2 \cup L_3) \tag{1}$$
$$L_1(L_2 \cap L_3) \subseteq L_1L_2 \cap L_1L_3 \tag{2}$$

Weshalb nicht '=' bei (2)?

Sei $\Sigma = \Sigma_{\text{bool}} = \{0, 1\}$, $L_1 = \{\lambda, 1\}$, $L_2 = \{0\}$ und $L_3 = \{10\}$.

Dann haben wir $L_1(L_2 \cap L_3) = \emptyset \neq \{10\} = L_1L_2 \cap L_1L_3$.

Beweise im Buch/Vorlesung

Homomorphismus

Seien Σ_1 und Σ_2 zwei beliebige Alphabete. Ein Homomorphismus von Σ_1^* nach Σ_2^* ist jede Funktion $h : \Sigma_1^* \to \Sigma_2^*$ mit den folgenden Eigenschaften:

(i) $h(\lambda) = \lambda$ und

(ii) $h(uv) = h(u) \cdot h(v)$ für alle $u, v \in \Sigma_1^*$.

Wir können Probleme etc. in anderen Alphabeten kodieren. So wie wir verschiedenste Konzepte, die wir auf Computer übertragen in Σ_{bool} kodieren.

2 Algorithmische Probleme

Mathematische Definition folgt in Kapitel 4 (Turingmaschinen).

Algorithmen - Provisorische Definition

Vorerst betrachten wir Programme, die **für jede zulässige Eingabe halten und eine Ausgabe liefern**, als Algorithmen.
Wir betrachten ein Programm (Algorithmus) A als Abbildung $A : \Sigma_1^* \to \Sigma_2^*$ für beliebige Alphabete Σ_1 und Σ_2. Dies bedeutet, dass

(i) die Eingaben als Wörter über Σ_1 kodiert sind,

(ii) die Ausgaben als Wörter über Σ_2 kodiert sind und

(iii) A für jede Eingabe eine eindeutige Ausgabe bestimmt.

A und B äquivalent $\iff$ Eingabealphabet Σ gleich, $A(x) = B(x), \forall x \in \Sigma^*$

Ie. diese Notion von "Äquivalenz" bezieht sich nur auf die Ein und Ausgabe.

Entscheidungsproblem

Das **Entscheidungsproblem** (Σ, L) für ein gegebenes Alphabet Σ und eine gegebene Sprache $L \subseteq \Sigma^*$ ist, für jedes $x \in \Sigma^*$ zu entscheiden, ob

$$x \in L \text{ oder } x \notin L.$$

Ein Algorithmus A **löst** das Entscheidungsproblem (Σ, L), falls für alle $x \in \Sigma^*$ gilt:

$$A(x) = \begin{cases} 1, & \text{falls } x \in L, \\ 0, & \text{falls } x \notin L. \end{cases}$$

Wir sagen auch, dass A die Sprache L erkennt.

Rekursive Sprachen

Wenn für eine Sprache L ein Algorithmus existiert, der L erkennt, sagen wir, dass L **rekursiv** ist.

Wir sind oft an spezifischen Eigenschaften von Wörtern aus Σ^* interessiert, die wir mit einer Sprache $L \subseteq \Sigma^*$ beschreiben können.

Dabei sind dann L die Wörter, die die Eigenschaft haben und $L^{\complement} = \Sigma^* \setminus L$ die Wörter, die diese Eigenschaft nicht haben.

Jetzt ist die allgemeine Formulierung von Vorteil!

i. **Primzahlen finden**:

   Entscheidungsproblem $(\Sigma_{\text{bool}}, L_p)$ wobei
   $L_p = \{x \in (\Sigma_{\text{bool}})^* \mid \text{Nummer}(x) \text{ ist prim}\}$.

ii. **Syntaktisch korrekte Programme**:

   Entscheidungsproblem $(\Sigma_{\text{Tastatur}}, L_{C++})$ wobei
   $L_{C++} = \{x \in (\Sigma_{\text{Tastatur}})^* \mid x \text{ ist ein syntaktisch korrektes C++ Programm}\}$.

iii. **Hamiltonkreise finden:**

   Entscheidungsproblem (Σ, HK) wobei $\Sigma = \{0, 1, \#\}$ und
   HK= $\{x \in \Sigma^* \mid x \text{ kodiert einen Graphen, der einen Hamiltonkreis enthält.}\}$

Äquivalenzprobleme $\subset$ Entscheidungsprobleme

Seien Σ und Γ zwei Alphabete.

- Wir sagen, dass ein Algorithmus A eine **Funktion (Transformation)** $f : \Sigma^* \to \Gamma^*$ **berechnet (realisiert)**, falls
$$A(x) = f(x) \text{ für alle } x \in \Sigma^*$$
- Sei $R \subseteq \Sigma^* \times \Gamma^*$ eine Relation in Σ^* und Γ^*. Ein Algorithmus A **berechnet** R (bzw. **löst das Relationsproblem** R), falls für jedes $x \in \Sigma^*$, für das ein $y \in \Gamma^*$ mit $(x, y) \in R$ existiert, gilt:
$$(x, A(x)) \in R$$

Optimierungsproblem

Ein **Optimierungsproblem** ist ein 6-Tupel $\mathcal{U} = (\Sigma_I, \Sigma_O, L, M, \text{cost}, \text{goal})$, wobei:

(i) Σ_I ist ein Alphabet (genannt **Eingabealphabet**),

(ii) Σ_O ist ein Alphabet (genannt **Ausgabealphabet**),

(iii) $L \subseteq \Sigma_I^*$ ist die Sprache der **zulässigen Eingaben** (als Eingaben kommen nur Wörter in Frage, die eine sinnvolle Bedeutung haben). Ein $x \in L$ wird ein **Problemfall (Instanz) von** $\mathcal{U}$ genannt.

(iv) M ist eine Funktion von L nach $\mathcal{P}(\Sigma_O^*)$, und für jedes $x \in L$ ist $M(x)$ die **Menge der zulässigen Lösungen für** x,

(v) **cost** ist eine Funktion, **cost**: $\bigcup_{x \in L}(\mathcal{M}(x) \times \{x\}) \to \mathbb{R}^+$, genannt **Kostenfunktion**,

(vi) **goal** $\in \{\text{Minimum}, \text{Maximum}\}$ ist das **Optimierungsziel**.

Eine zulässige Lösung $\alpha \in \mathcal{M}(x)$ heisst **optimal** für den Problemfall x des Optimierungsproblems $\mathcal{U}$, falls

$$\text{cost}(\alpha, x) = \mathbf{Opt}_{\mathcal{U}}(x) = \text{ goal}\{\text{cost}(\beta, x) \mid \beta \mathcal{M}(x)\}.$$

Ein Algorithmus A **löst** $\mathcal{U}$, falls für jedes $x \in L$

(i) $A(x) \in \mathcal{M}(x)$

(ii) $\text{cost}(A(x), x) = \text{ goal}\{\text{cost}(\beta, x) \mid \beta \in \mathcal{M}(x)\}$.

3 Kolmogorov Komplexität

3.1 Theorie

Algorithmen generieren Wörter

Sei Σ ein Alphabet und $x \in \Sigma^*$. Wir sagen, dass ein Algorithmus A das Wort x **generiert**, falls A für die Eingabe λ die Ausgabe x liefert.

Beispiel:

```
A_n:  begin
          for i = 1 to n;
              write(01);
      end
```

A_n generiert $(01)^n$.

Aufzählungsalgorithmus

Sei Σ ein Alphabet und sei $L \subseteq \Sigma^*$. A ist ein **Aufzählungsalgorithmus für** L, falls A für jede Eingabe $n \in \mathbb{N} \setminus \{0\}$ die Wortfolge $x_1, ..., x_n$ ausgibt, wobei $x_1, ..., x_n$ die kanonisch n ersten Wörter in L sind.

Aufgabe 2.21

Beweisen Sie, dass eine Sprache L genau dann rekursiv ist, wenn ein Aufzählungsalgorithmus für L existiert.

Das **Entscheidungsproblem** (Σ, L) für ein gegebenes Alphabet Σ und eine gegebene Sprache $L \subseteq \Sigma^*$ ist, für jedes $x \in \Sigma^*$ zu entscheiden, ob

$$x \in L \text{ oder } x \notin L.$$

Ein Algorithmus A **löst** das Entscheidungsproblem (Σ, L), falls für alle $x \in \Sigma^*$ gilt:

$$A(x) = \begin{cases} 1, & \text{falls } x \in L, \\ 0, & \text{falls } x \notin L. \end{cases}$$

Wir sagen auch, dass A die Sprache L erkennt.

L rekursiv ($\Longrightarrow$) existiert Aufzählungsalgorithmus:

Sei A ein Algorithmus, der L erkennt. Wir beschreiben nun einen Aufzählungsalgorithmus B konstruktiv.

Algorithm 1 $B(\Sigma, n)$

```
i ← 0
while i ≤ n do
    w ← kanonisch nächstes Wort über Σ*
    if A(w) = 1 then
        print(w)
        i ← i + 1
    end if
end while
```

Aufzählungsalgorithmus $B \implies L$ rekursiv:

Algorithm 2 $A(\Sigma, w)$

```
n ← |Σ|^(|w|+1)
L ← B(Σ, n)
if w ∈ L then
    print(1)
else
    print(0)
end if
```

Es gibt ein kleines Problem. B könnte unendlich lange laufen, falls $n > |L|$.
Es sollte nicht so schwierig sein, B zu modifizieren, dass es die Berechnung aufhört, falls es keine weiteren Wörter in L gibt.

Information messen Wir beschränken uns auf Σ_{bool}

Kolmogorov-Komplexität

Für jedes Wort $x \in (\Sigma_{\text{bool}})^*$ ist die **Kolmogorov-Komplexität $K(x)$ des Wortes** x das Minimum der binären Längen, der Pascal-Programme, die x generieren.

$K(x)$ ist die kürzestmögliche Länge einer Beschreibung von x.

Die einfachste (und triviale) Beschreibung von x, ist wenn man x direkt angibt.

x kann aber eine Struktur oder Regelmässigkeit haben, die eine Komprimierung erlaubt.

Welche Programmiersprache gewählt wird verändert die Kolmogorov-Komplexität nur um eine Konstante. (Satz 2.1)

Beispiel

Sei $w = 01010101010101010101010101010101010101$. Die Länge von w ist $|w| = 40$ und die triviale Beschreibunglänge wäre wie gegeben 40.

Aber durch die Regelmässigkeit von einer 20-fachen Wiederholung der Sequenz 01, können w auch durch $(01)^{20}$ beschreiben. Hierbei ist die Beschreibungslänge ein wenig mehr als 4 Zeichen.

Grundlegende Resultate

Es existiert eine Konstante d, so dass für jedes $x \in (\Sigma_{\text{bool}})^*$

$$K(x) \leq |x| + d$$

Die **Kolmogorov-Komplexität einer natürlichen Zahl** n ist $K(n) = K(\text{Bin}(n))$.

Lemma 2.5 - Nichtkomprimierbar

Für jede Zahl $n \in \mathbb{N} \setminus \{0\}$ existiert ein Wort $w_n \in (\Sigma_{\text{bool}})^n$, so dass

$$K(w_n) \geq |w_n| = n$$

Beweis

Es gibt 2^n Wörter $x_1, ..., x_{2^n}$ über Σ_{bool} der Länge n. Wir bezeichnen $C(x_i)$ als den Bitstring des kürzesten Programms, der x_i generieren kann. Es ist klar, dass für $i \neq j : C(x_i) \neq C(x_j)$.

Die Anzahl der nichtleeren Bitstrings, i.e. der Wörter der Länge $< n$ über Σ_{bool} ist:

$$\sum_{i=1}^{n-1} 2^i = 2^n - 2 < 2^n$$

Also muss es unter den Wörtern $x_1, ..., x_{2^n}$ mindestens ein Wort x_k mit $K(x_k) \geq n$ geben.

■

Satz 2.1 - Programmiersprachen

Für jedes Wort $x \in (\Sigma_{\text{bool}})^*$ und jede Programmiersprache A sei $K_A(x)$ die Kolmogorov-Komplexität von x bezüglich der Programmiersprache A.

Seien A und B Programmiersprachen. Es existiert eine Konstante $c_{A,B}$, die nur von A und B abhängt, so dass

$$|K_A(x) - K_B(x)| \leq c_{A,B}$$

für alle $x \in (\Sigma_{\text{bool}})^*$.

Beweis im Buch/Vorlesung

Ein zufälliges Wort

Ein Wort $x \in (\Sigma_{\text{bool}})^*$ heisst **zufällig**, falls $K(x) \geq |x|$.
Eine Zahl n heisst **zufällig**, falls $K(n) = K(\text{Bin}(n)) \geq \lceil \log_2(n+1) \rceil - 1$.

Jede Binär-Darstellung beginnt immer mit einer 1, deshalb können wir die Länge der Binär-Darstellung um 1 verkürzen.

Zufälligkeit hier bedeutet, dass ein Wort völlig unstrukturiert ist und sich nicht komprimieren lässt. Es hat nichts mit Wahrscheinlichkeit zu tun.

Satz 2.2

Sei L eine Sprache über Σ_{bool}. Sei für jedes $n \in \mathbb{N} \setminus \{0\}$, z_n das n-te Wort in L bezüglich der kanonischen Ordnung. Wenn ein Programm A_L existiert, das das Entscheidungsproblem $(\Sigma_{\text{bool}}, L)$ löst, dann gilt für alle $n \in \mathbb{N} \setminus \{0\}$, dass

$$K(z_n) \leq \lceil \log_2(n+1) \rceil + c$$

wobei c eine von n unabhängige Konstante ist.

Beweisidee

Wir können aus A_L, ein Programm entwerfen, dass das kanonisch n-te Wort generiert, indem wir in der kanonischen Reihenfolge alle Wörter $x \in (\Sigma_{bool})^*$ durchgehen und mit A_L entscheiden, ob $x \in L$. Dann können wir einen Counter c haben und den Prozess abbrechen, wenn der Counter $c = n$ wird und dann dieses Wort ausgeben.

Wir sehen, dass dieses Programm ausser der Eingabe n immer gleich ist. Sei die Länge dieses Programms c, dann können wir für das n-te Wort der Sprache L, z_n, die Kolmogorov-Komplexität auf n reduzieren, bzw:

$$K(z_n) \leq \lceil \log_2(n+1) \rceil + c$$

■

Primzahlsatz

Für jede positive ganz Zahl n sei $\text{Prim}(n)$ die Anzahl der Primzahlen kleiner gleich n.

$$\lim_{n \to \infty} \frac{\text{Prim}(n)}{n/\ln n} = 1$$

Nützliche Ungleichung

$$\ln n - \frac{3}{2} < \frac{n}{\text{Prim(n)}} < \ln n - \frac{1}{2}$$

für alle $n \geq 67$.

Lemma 2.6 - schwache Version des Primzahlsatzes

Sei $n_1, n_2, n_3, ...$ eine steigende unendliche Folge natürlicher Zahlen mit $K(n_i) \geq \lceil \log_2 n_i \rceil / 2$. Für jedes $i \in \mathbb{N} \setminus \{0\}$ sei q_i die grösste Primzahl, die die Zahl n_i teilt. Dann ist die Menge $Q = \{q_i \mid i \in \mathbb{N} \setminus \{0\}\}$ unendlich.

Beweis: Wir beweisen diese Aussage per Widerspruch:

Nehmen wir zum Widerspruch an, dass die Menge $Q = \{q_i \mid i \in \mathbb{N} \setminus \{0\}\}$ sei endlich.

Sei q_m die grösste Primzahl in Q. Dann können wir jede Zahl n_i eindeutig als

$$n_i = q_1^{r_{i,1}} \cdot q_2^{r_{i,2}} \cdot \dots \cdot q_m^{r_{i,m}}$$

für irgendwelche $r_{i,1}, r_{i,2}, ..., r_{i,m} \in \mathbb{N}$ darstellen. Sei c die binäre Länge eines Programms, dass diese $r_{i,j}$ als Eingaben nimmt und n_i erzeugt (A ist für alle $i \in \mathbb{N}$ bis auf die Eingaben $r_{i,1}, ..., r_{i,m}$ gleich).

Dann gilt:

$$K(n_i) \leq c + 8 \cdot (\lceil \log_2(r_{i,1} + 1) \rceil + \lceil \log_2(r_{i,2} + 1) \rceil + ... + \lceil \log_2(r_{i,m} + 1) \rceil)$$

Die multiplikative Konstante 8 kommt daher, dass wir für die Zahlen $r_{i,1}, r_{i,2}, ..., r_{i,m}$ dieselbe Kodierung, wie für den Rest des Programmes verwenden (z.B. ASCII-Kodierung), damit ihre Darstellungen eindeutig voneinander getrennt werden können. Weil $r_{i,j} \leq \log_2 n_i, \forall j \in \{1, ..., m\}$ erhalten wir

$$K(n_i) \leq c + 8m \cdot \lceil \log_2(\log_2 n_i + 1) \rceil, \forall i \in \mathbb{N} \setminus \{0\}$$

Weil m und c Konstanten unabhängig von i sind, kann

$$\lceil \log_2 n_i \rceil / 2 \leq K(n_i) \leq c + 8m \cdot \lceil \log_2(\log_2 n_i + 1) \rceil$$

$$\lceil \log_2 n_i \rceil / 2 \leq c + 8m \cdot \lceil \log_2(\log_2 n_i + 1) \rceil$$

nur für endlich viele $i \in \mathbb{N} \setminus \{0\}$ gelten.

Dies ist ein Widerspruch!

Folglich ist die Menge Q unendlich.

■

3.2 How To Kolmogorov

Aufgabentyp 1

Sei $w_n = (010)^{3^{2n^3}} \in \{0,1\}^*$ für alle $n \in \mathbb{N} \setminus \{0\}$. Gib eine möglichst gute obere Schranke für die Kolmogorov-Komplexität von w_n an, gemessen in der Länge von w_n.

Lösung Typ 1

Wir zeigen ein Programm, dass n als Eingabe nimmt und w_n druckt:

```
W_n:    begin
        M := n;
        M := 2 × M × M × M;
        J := 1;
        for I = 1 to M
            J := J × 3;
        for I = 1 to J;
            write(010);
    end
```

Der einzige variable Teil dieses Algorithmus ist n. Der restliche Code ist von konstanter Länge. Die binäre Länge dieses Programms kann von oben durch

$$\lceil \log_2(n + 1) \rceil + c$$

beschränkt werden, für eine Konstante c.

Somit folgt

$$K(w_n) \leq \log_2(n) + c'$$

Wir berechnen die Länge von w_n als $|w_n| = |010| \cdot 3^{2n^3} = 3^{2n^3+1}$.

Mit ein wenig umrechnen erhalten wir

$$n = \sqrt[3]{\frac{\log_3 |w_n| - 1}{2}}$$

und die obere Schranke

$$K(w_n) \leq \log_2 \left(\sqrt[3]{\frac{\log_3 |w_n| - 1}{2}} \right) + c' \leq \log_2 \log_3 |w_n| + c''$$

Aufgabentyp 2

Geben Sie eine unendliche Folge von Wörtern $y_1 < y_2 < ...$ an, so dass eine Konstante $c \in \mathbb{N}$ existiert, so dass für alle $i \geq 1$

$$K(y_i) \leq \log_2 \log_2 \log_3 \log_2(|y_i|) + c$$

Lösung Typ 2

Wir definieren die Folge $y_1, y_2, ...$ mit $y_i = 0^{2^{3^{2^i}}}$ für alle $i \in \mathbb{N}$. Da $|y_i| < |y_{i+1}|$ folgt die geforderte Ordnung.

Es gilt

$$i = \log_2 \log_3 \log_2 |y_i| \text{ für } i \geq 1$$

Wir zeigen ein Programm, dass i als Eingabe nimmt und y_i druckt:

```
begin
    M := i;
    M := 2^(3^(2^M));
    for I = 1 to M;
        write(010);
end
```

Das ^ für die Exponentiation ist nicht Teil der originalen Pascal Syntax, aber wir verwenden es um unser Programm lesbarer zu machen.

Der einzige variable Teil dieses Programms ist das i. Der Rest hat konstante Länge. Demnach kann die Länge diese Programms für eine Konstante c' durch

$$\lceil \log_2(i+1) \rceil + c'$$

von oben beschränkt werden.

Somit folgt

$$\begin{aligned} K(y_i) &\leq \log_2(i) + c \\ &\leq \log_2 \log_2 \log_3 \log_2 |y_i| + c \end{aligned}$$

für eine Konstante c.

Aufgabentyp 3

Sei $M = \{7^i \mid i \in \mathbb{N},\ i \leq 2^n - 1\}$. Beweisen Sie, dass mindestens sieben Achtel der Zahlen in M Kolmogorov-Komplexität von mindestens $n - 3$ haben.

Lösung Typ 3

Wir zeigen, dass höchstens $\frac{1}{8}$ der Zahlen $x \in M$ eine Kolmogorov-Komplexität $K(x) \leq n-4$ haben.

Nehmen wir zum Widerspruch an, dass M mehr als $\frac{1}{8}|M|$ Zahlen x enthält, mit $K(x) \leq n-4$.

Die Programme, die diese Wörter generieren, müssen paarweise verschieden sein, da die Wörter paarweise verschieden sind.

Es gibt aber höchstens

$$\sum_{k=0}^{n-4} 2^k = 2^{n-3} - 1 < \frac{1}{8}|M|$$

Bitstrings mit Länge $\leq n-4$. **Widerspruch.**

4 Endliche Automaten - Einführung

4.1 Erster Ansatz zur Modellierung von Algorithmen

Ein (deterministischer) **endlicher Automat (EA)** ist ein Quintupel $M = (Q, \Sigma, \delta, q_0, F)$, wobei

(i) Q eine endliche Menge von **Zuständen** ist,

(ii) Σ ein Alphabet, genannt **Eingabealphabet**, ist,

(iii) $q_0 \in Q$ der Anfangszustand ist,

(iv) $F \subseteq Q$ die **Menge der akzeptierenden Zustände** ist und

(v) $\delta : Q \times \Sigma \to Q$ die **Übergangsfunktion** ist.

Konfigurationen

Eine **Konfiguration** von M ist ein Tupel $(q, w) \in Q \times \Sigma^*$.

- "M befindet sich in einer Konfiguration $(q, w) \in Q \times \Sigma^*$, wenn M im Zustand q ist und noch das Suffix w eines Eingabewortes lesen soll."
- Die Konfiguration $(q_0, x) \in \{q_0\} \times \Sigma^*$ heisst die **Startkonfiguration von M auf x**.
- Jede Konfiguration aus $Q \times \{\lambda\}$ nennt man **Endkonfiguration**.

Ein **Schritt** von M ist eine Relation (auf Konfigurationen) $\vdash_M \subseteq (Q \times \Sigma^*) \times (Q \times \Sigma^*)$, definiert durch

$$(q, w) \vdash_M (p, x) \iff w = ax, a \in \Sigma \text{ und } \delta(q, a) = p.$$

Berechnungen

Eine **Berechnung** C von M ist eine endliche Folge $C = C_0, C_1, ..., C_n$ von Konfigurationen, so dass

$$C_i \vdash_M C_{i+1} \text{ für alle } 0 \leq i \leq n-1.$$

C ist die **Berechnung von M auf einer Eingabe** $x \in \Sigma^*$, falls $C_0 = (q_0, x)$ und $C_n \in Q \times \{\lambda\}$ eine Endkonfiguration ist.

Falls $C_n \in F \times \{\lambda\}$, sagen wir, dass C eine **akzeptierende Berechnung** von M auf x ist, und dass M **das Wort x akzeptiert**.

Falls $C_n \in (Q \setminus F) \times \{\lambda\}$, sagen wir, dass C eine **verwerfende Berechnung** von M auf x ist, und dass M **das Wort** x **verwirft (nicht akzeptiert)**.

Transitivität von $\vdash_M$ und δ

Sei $M = (Q, \Sigma, \delta, q_0, F)$ ein endlicher Automat. Wir definieren $\vdash_M^*$ als die reflexive und transitive Hülle der Schrittrelation $\vdash_M$ von M; daher ist

$$(q, w) \vdash_M^* (p, u) \iff (q = p \wedge w = u) \text{ oder } \exists k \in \mathbb{N} \setminus \{0\},$$

so dass

(i) $w = a_1 a_2 ... a_k u, a_i \in \Sigma$ für $i = 1, 2, ..., k$, und

(ii) $\exists r_1, r_2, ..., r_{k-1} \in Q$, so dass

$$(q, w) \vdash_M (r_1, a_2 ... a_k u) \vdash_M ... \vdash_M (r_{k-1}, a_k u) \vdash_M (p, u)$$

Wir definieren $\hat{\delta} : Q \times \Sigma^* \to Q$ durch:

(i) $\hat{\delta}(q, \lambda) = q$ für alle $q \in Q$ und

(ii) $\hat{\delta}(q, wa) = \delta(\hat{\delta}(q, w), a)$ für alle $a \in \Sigma, w \in \Sigma^*, q \in Q$.

$$\hat{\delta}(q, w) = p \iff (q, w) \vdash_M^* (p, \lambda)$$

4.2 Reguläre Sprachen

Die **von** M **akzeptierte Sprache** $L(M)$ ist definiert als

$$\begin{aligned} L(M) &= \{w \in \Sigma^* \mid \text{Berechnung von } M \text{ auf } w \text{ endet in } (p, \lambda) \in F \times \{\lambda\}\} \\ &= \{w \in \Sigma^* \mid (q_0, w) \vdash_M^* (p, \lambda) \wedge p \in F\} \\ &= \{w \in \Sigma^* \mid \hat{\delta}(q_0, w) \in F\} \end{aligned}$$

$\mathcal{L}_{\mathbf{EA}} = \{L(M) \mid M \text{ ist ein EA}\}$ ist die Klasse der Sprachen, die von endlichen Automaten akzeptiert werden.
$\mathcal{L}_{\text{EA}}$ bezeichnet man auch als die **Klasse der regulären Sprachen**, und jede Sprache $L \in \mathcal{L}_{\text{EA}}$ wird **regulär** genannt.

Klassen für alle Zustände im Endlichen Automaten

Für alle $p \in Q$ definieren wir die Klasse

$$\begin{aligned} \mathbf{Kl[p]} &= \{w \in \Sigma^* \mid \hat{\delta}(q_0, w) = p\} \\ &= \{w \in \Sigma^* \mid (q_0, w) \vdash_M^* (p, \lambda)\} \end{aligned}$$

Wir bemerken dann

$$\bigcup_{q \in Q} \text{Kl[q]} = \Sigma^*$$

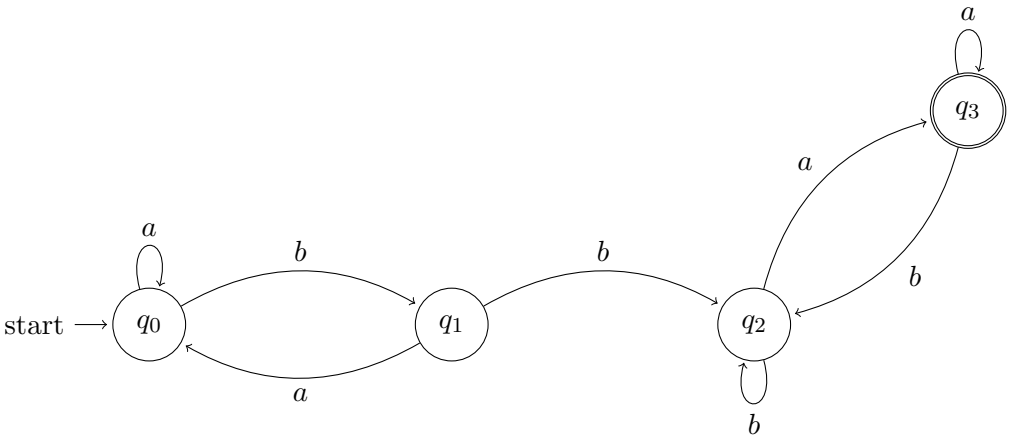
$$\text{Kl}[q] \cap \text{Kl}[p] = \emptyset, \forall p, q \in Q, p \neq q$$

$$L(M) = \bigcup_{q \in F} \text{Kl}[q]$$

EA Konstruktion - Beispielaufgabe

Entwerfen sie für folgende Sprache einen Endlichen Automat und geben Sie eine Beschreibung von Kl[q] für jeden Zustand $q \in Q$.

$$L_1 = \{xbbya \in \{a,b\}^* \mid x, y \in \{a,b\}^*\}$$



Wir beschreiben nun die Klassen für die Zustände q_0, q_1, q_2, q_3:
Kl[q_0] = $\{wa \in \{a,b\}^* \mid$ Das Wort w enthält nicht die Teilfolge $bb\} \cup \{\lambda\}$
Kl[q_1] = $\{wb \in \{a,b\}^* \mid$ Das Wort w enthält nicht die Teilfolge $bb\}$
Kl[q_3] = $\{wa \in \{a,b\}^* \mid$ Das Wort w enthält die Teilfolge $bb\} = L_1$
Kl[q_2] = $\{a,b\}^* - (\text{Kl}[q_0] \cup \text{Kl}[q_1] \cup \text{Kl}[q_3])$

4.3 Produktautomaten - Simulationen

Lemma 3.2

Sei Σ ein Alphabet und seien $M_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$ und $M_2 = (Q_2, \Sigma, \delta_2, q_{02}, F_2)$ zwei EA. Für jede Mengenoperation $\odot \in \{\cup, \cap, -\}$ existiert ein EA M, so dass

$$L(M) = L(M_1) \odot L(M_2).$$

Sei $M = (Q, \Sigma, \delta, q_0, F_\odot)$, wobei

(i) $Q = Q_1 \times Q_2$

(ii) $q_0 = (q_{01}, q_{02})$

(iii) für alle $q \in Q_1$, $p \in Q_2$ und $a \in \Sigma$, $\delta((q,p),a) = (\delta_1(q,a), \delta_2(p,a))$,

(iv) falls $\odot = \cup$, dann ist $F = F_1 \times Q_2 \cup Q_1 \times F_2$
falls $\odot = \cap$, dann ist $F = F_1 \times F_2$, und
falls $\odot = -$, dann ist $F = F_1 \times (Q_2 - F_2)$.

Produktautomat - Beispielaufgabe

Verwenden Sie die Methode des modularen Entwurfs (Konstruktion eines Produktautomaten), um einen endlichen Automaten (in Diagrammdarstellung) für die Sprache

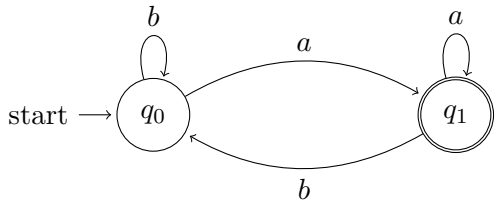
$$L = \{w \in \{a,b\}^* \mid |w|_a = 2 \text{ oder } w = ya\}$$

zu entwerfen. Zeichnen Sie auch jeden der Teilautomaten und geben Sie für die Teilautomaten für jeden Zustand q die Klasse Kl[q] an.

Wir teilen L wie folgt auf:

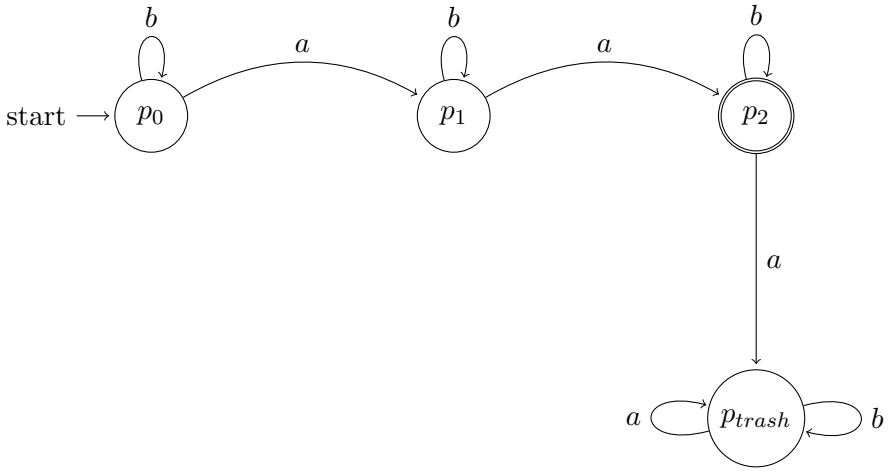
$$L = L_1 \cup L_2 \text{ wobei gilt:}$$
$$L_1 = \{w \in \{a,b\}^* \mid w = ya\}$$
$$L_2 = \{w \in \{a,b\}^* \mid |w|_a = 2\}$$

Zuerst zeichnen wir die 2 einzelnen Teilautomaten und geben für jeden Zustand q bzw. p die Klasse Kl[q] respektive Kl[p] an:



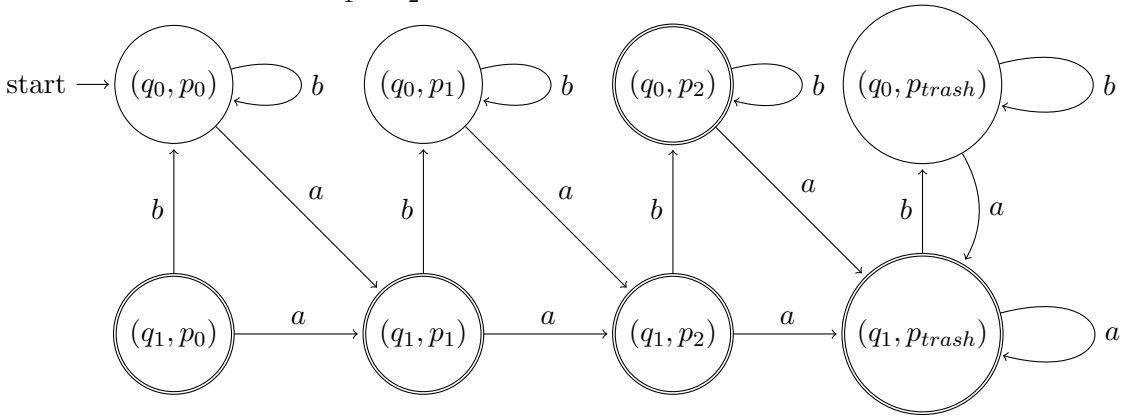
erster Teilautomat: $L_1 = \{w \in \{a,b\}^* \mid w = ya\}$
Wir beschreiben nun die Zustände für die Klassen q_0 und q_1:
Kl[q_0] = $\{yb \mid y \in \{a,b\}^*\} \cup \{\lambda\}$
Kl[q_1] = $\{ya \mid y \in \{a,b\}^*\}$

zweiter Teilautomat: $L_2 = \{w \in \{a,b\}^* \mid |w|_a = 2\}$



Wir beschreiben nun die Zustände für die Klassen p_0, p_1, p_2, p_{trash}: Kl[p_0] = $\{w \in \{a,b\}^* \mid |w|_a = 0\}$
Kl[p_1] = $\{w \in \{a,b\}^* \mid |w|_a = 1\}$
Kl[p_2] = $\{w \in \{a,b\}^* \mid |w|_a = 2\}$
Kl[p_{trash}] = $\{w \in \{a,b\}^* \mid |w|_a > 2\}$

Zum Schluss kombinieren wir diese Teilautomaten zu einem Produktautomaten:
Produktautomat: $L = L_1 \cup L_2$



5 Beweise für Nichtregularität

5.1 Einführung und grundlegende Tipps

i. Wichtiges Unterkapitel. Kommt fast garantiert am Midterm.

ii. Um $L \notin \mathcal{L}_{\text{EA}}$ zu zeigen, genügt es zu beweisen, dass es keinen EA gibt, der L akzeptiert.

iii. Nichtexistenz ist generell sehr schwer zu beweisen, da aber die Klasse der endlichen Automaten sehr eingeschränkt ist, ist dies nicht so schwierig.

iv. Wir führen Widerspruchsbeweise.

v. Es gibt 3 Arten Nichtregularitätsbeweise zu führen (Lemma 3.3, Pumping-Lemma und Kolmogorov-Komplexität).

vi. Ihr müsst alle 3 Methoden können. Ist aber halb so wild.

5.2 Theorie für Nichtregularitätsbeweise

5.2.1 Lemma 3.3 Methode

Lemma 3.3

Sei $A = (Q, \Sigma, \delta_A, q_0, F)$ ein EA. Seien $x, y \in \Sigma^*, x \neq y$, so dass

$$\hat{\delta}_A(q_0, x) = p = \hat{\delta}_A(q_0, y)$$

für ein $p \in Q$ (also $x, y \in \text{Kl}[p]$). Dann existiert für jedes $z \in \Sigma^*$ ein $r \in Q$, so dass xz und $yz \in \text{Kl}[r]$, also gilt insbesondere

$$xz \in L(A) \iff yz \in L(A)$$

Beweis:

Aus der Existenz der Berechnungen

$(q_0, x) \vdash_A^* (p, \lambda)$ und $(q_0, y) \vdash_A^* (p, \lambda)$ von A folgt die Existenz der Berechnungen auf xz und yz:

$(q_0, xz) \vdash_A^* (p, z)$ und $(q_0, yz) \vdash_A^* (p, z)$ für alle $z \in \Sigma^*$.

Wenn $r = \hat{\delta}_A(p, z)$ ist, dann ist die Berechnung von A auf xz und yz:

$(q_0, xz) \vdash_A^* (p, z) \vdash_A^* (r, \lambda)$ und $(q_0, yz) \vdash_A^* (p, z) \vdash_A^* (r, \lambda)$.

Wenn $r \in F$, dann sind beide Wörter xz und yz in $L(A)$. Falls $r \notin F$, dann sind $xz, yz \notin L(A)$.

■

Bemerkungen

- Von den 3 vorgestellten Methoden, ist diese Methode die einzige, die (unter der richtigen Anwendung) garantiert für jede nichtreguläre Sprache funktioniert.
- Um die Nichtregularität von L zu beweisen, verwenden wir die Endlichkeit von Q und das Pigeonhole-Principle.

Beispielaufgabe - Lemma 3.3

Betrachten wir mal eine Beispielaufgabe mit dieser Methode am Paradebeispiel

$$L = \{0^n 1^n \mid n \in \mathbb{N}\}$$

Nehmen wir zum Widerspruch an L sei regulär.

Dann existiert ein EA $A = (Q, \Sigma, \delta, q_0, F)$ mit $L(A) = L$.

Wir betrachten die Wörter $0^1, \ldots, 0^{|Q|+1}$. Per Pigeonhole-Principle existiert O.B.d.A. $i < j$, so dass

$$\hat{\delta}(q_0, 0^i) = \hat{\delta}(q_0, 0^j)$$

Nach Lemma 3.3 gilt

$$0^i z \in L \iff 0^j z \in L$$

für alle $z \in (\Sigma_{\text{bool}})^*$. Dies führt aber zu einem Widerspruch, weil für $z = 1^i$ das Wort $0^i 1^i \in L$ aber $0^j 1^i \notin L$.

5.2.2 Pumping Lemma Methode

Pumping Lemma

Sei L regulär. Dann existiert eine Konstante $n_0 \in \mathbb{N}$, so dass jedes Wort $w \in \Sigma^*$ mit $|w| \geq n_0$ in drei Teile x, y und z zerlegen lässt, das heisst $w = xyz$, wobei

(i) $|yx| \leq n_0$

(ii) $|x| \geq 1$

(iii) entweder $\{yx^k z \mid k \in \mathbb{N}\} \subseteq L$ oder $\{yx^k z \mid k \in \mathbb{N}\} \cap L = \emptyset$.

Beweis

Sei $L \in \Sigma^*$ regulär. Dann existiert ein EA $A = (Q, \Sigma, \delta_A, q_0, F)$, so dass $L(A) = L$.

Sei $n_0 = |Q|$ und $w \in \Sigma^*$ mit $|w| \geq n_0$. Dann ist $w = w_1 w_2 ... w_{n_0} u$, wobei $w_i \in \Sigma$ für $i = 1, ..., n_0$ und $u \in \Sigma^*$. Betrachten wir die Berechnung auf $w_1 w_2 ... w_{n_0}$:

$$(q_0, w_1 w_2 w_3 ... w_{n_0}) \vdash_A (q_1, w_2 w_3 ... w_{n_0}) \vdash_A ... \vdash_A (q_{n_0-1}, w_{n_0}) \vdash_A (q_{n_0}, \lambda)$$

In dieser Berechnung kommen n_0+1 Zustände $q_0, q_1, ..., q_{n_0}$ vor. Da $|Q| = n_0$, existieren $i, j \in \{0, 1, ..., n_0\}, i < j$, so dass $q_i = q_j$. Daher haben wir in der Berechnung die Konfigurationen

$$(q_0, w_1 w_2 w_3 ... w_{n_0}) \vdash_A^* (q_i, w_{i+1} w_{i+2} ... w_{n_0}) \vdash_A^* (q_i, w_{j+1} ... w_{n_0}) \vdash_A^* (q_{n_0}, \lambda)$$

Dies impliziert

$$(q_i, w_{i+1} w_{i+2} ... w_j) \vdash_A^* (q_i, \lambda) \qquad (1)$$

Wir setzen nun $y = w_1 ... w_i$, $x = w_{i+1} ... w_j$ und $z = w_{j+1} ... w_{n_0} u$, so dass $w = yxz$.

Wir überprüfen nun die Eigenschaften (i),(ii) und (iii):

(i) $yx = w_1 ... w_i w_{i+1} ... w_j$ und daher $|yx| = j \leq n_0$.

(ii) Da $|x| \geq j - i$ und $i < j$, ist $|x| \geq 1$.

(iii) (1) impliziert $(q_i, x^k) \vdash_A^* (q_i, \lambda)$ für alle $k \in \mathbb{N}$. Folglich gilt für alle $k \in \mathbb{N}$:

$$(q_0, yx^k z) \vdash_A^* (q_i, x^k z) \vdash_A^* (q_i, z) \vdash_A^* (\hat{\delta}_A(q_i, z), \lambda)$$

Wir sehen, dass für alle $k \in \mathbb{N}$ die Berechnungen im gleichen Zustand $q_{end} = \hat{\delta}_A(q_i, z)$ enden. Falls also $q_{end} \in F$, akzeptiert A alle Wörter aus $\{yx^k z \mid k \in \mathbb{N}\}$. Falls $q_{end} \notin F$, dann akzeptiert A kein Wort aus $\{yx^k z \mid k \in \mathbb{N}\}$.

■

Beispielaufgabe - Pumping Lemma

Versuchen wir zu beweisen, dass

$$L_2 = \{wabw^{\mathbf{R}} \mid w \in \{a,b\}^*\}$$

nicht regulär ist.

Wir nehmen zum Widerspruch an, dass L_2 regulär ist.

Das Pumping-Lemma (Lemma 3.4) besagt, dass dann eine Konstante $n_0 \in \mathbb{N}$ existiert, so dass sich jedes Wort $w \in \Sigma*$ mit $|w| \geq n_0$ in drei Teile y, x, und z zerlegen lässt. ($\implies w = yxz$). Wobei folgendes gelten muss:

(i) $|yx| \leq n_0$

(ii) $|x| \geq 1$

(iii) **entweder** $\{yx^kz \mid k \in \mathbb{N}\} \subseteq L_2$ **oder** $\{yx^kz \mid k \in \mathbb{N}\} \cap L_2 = \emptyset$

Wir wählen $w = a^{n_0}aba^{n_0}$. Es ist leicht zu sehen das $|w| = 2n_0 + 2 \geq n_0$.

Da nach (i), $|yx| \leq n_0$ gelten muss, haben wir $y = a^l$ und $x = a^m$ für beliebige $l, m \in \mathbb{N}, l + m \leq n_0$.
Somit gilt $z = a^{n_0-(l+m)}aba^{n_0}$

Nach (ii) ist $m \geq 1$.

Wir haben also $\{yx^kz \mid k \in \mathbb{N}\} = \{a^{n_0-m+km}aba^{n_0} \mid k \in \mathbb{N}\}$

Da $yx^1z = a^{n_0}aba^{n_0}$ und

$a^{n_0}aba^{n_0} \in \{a^{n_0-m+km}aba^{n_0} \mid k \in \mathbb{N}\} \wedge a^{n_0}aba^{n_0} \in L_2$ gilt, folgt

$$\{a^{n_0-m+km}aba^{n_0} \mid k \in \mathbb{N}\} \cap L_2 \neq \emptyset$$

Wenn wir nun $k = 0$ wählen und uns daran erinnern, dass $m \geq 1$, erhalten wir folgendes

$$\Rightarrow yx^0z = yz = a^{n_0-m}aba^{n_0} \notin L_2$$

Daraus folgt,

$$\{a^{n_0-m+km}aba^{n_0} \mid k \in \mathbb{N}\} \not\subseteq L_2$$

Somit gilt (iii) nicht.

Dies ist ein Widerspruch! Somit haben wir gezeigt, dass die Sprache $L_2 = \{wabw^{\mathbf{R}} \mid w \in \{a,b\}^*\}$ nicht regulär ist.

5.2.3 Kolmogorov Methode

Satz 3.1

Sei $L \subseteq (\Sigma_{\text{bool}})^*$ eine reguläre Sprache. Sei $L_x = \{y \in (\Sigma_{\text{bool}})^* \mid xy \in L\}$ für jedes $x \in (\Sigma_{\text{bool}})^*$. Dann existiert eine Konstante **const**, so dass für alle $x, y \in (\Sigma_{\text{bool}})^*$

$$K(y) \leq \lceil \log_2(n+1) \rceil + \ \textbf{const},$$

falls y das n-te Wort in der Sprache L_x ist.

Wie wir sehen werden, beruht der Nichtregularitätsbeweis darauf, dass die Differenz von $|w_{n+1}| - |w_n|$ für kanonische Wörter $(w_i)_{i\in\mathbb{N}}$ beliebig gross werden kann.

Beispielaufgabe - Kolmogorov Methode

Verwenden Sie die Methode der Kolmogorov-Komplexität, um zu zeigen, dass die Sprache

$$L_1 = \{0^{n^2 \cdot 2^n} \mid n \in \mathbb{N}\}$$

nicht regulär ist.

Angenommen L_1 sei regulär.

Wir betrachten

$$L_{0^{m^2 \cdot 2^m + 1}} = \{y \mid 0^{m^2 \cdot 2^m + 1} y \in L_1\}.$$

Da

$$\begin{aligned}(m+1)^2 \cdot 2^{m+1} &= (m^2 + 2m + 1) \cdot 2^{m+1} \\ &= m^2 \cdot 2^m + m^2 \cdot 2^m + (2m+1) \cdot 2^{m+1} \\ &= m^2 \cdot 2^m + (m^2 + 4m + 2) \cdot 2^m\end{aligned}$$

ist für jedes $m \in \mathbb{N}$ das Wort $y_1 = 0^{(m^2+4m+2) \cdot 2^m - 1}$ das kanonisch erste Wort der Sprache $L_{0^{m^2 \cdot 2^m + 1}}$.

Nach Satz 3.1 existiert eine Konstante c, unabhängig von m, so dass

$$K(y_1) \leq \lceil \log_2(1+1) \rceil + c = 1 + c.$$

Die Anzahl aller Programme, deren Länge kleiner oder gleich $1 + c$ sind, ist endlich.

Da es aber unendlich viel Wörter der Form $0^{(m^2+4m+2) \cdot 2^m - 1}$ gibt, ist dies ein Widerspruch.

Demzufolge ist L_1 nicht regulär.

■

5.3 Weitere Aufgaben

Beispielaufgabe 1 - Direkte Methode (Lemma 3.3)

Verwende eine direkte Argumentation über den Automaten (unter Verwendung von Lemma 3.3), um zu zeigen, dass die Sprache

$$L_2 = \{w \in \{0,1\}^* \mid |u|_0 \leq |u|_1 \text{ für alle Präfixe } u \text{ von } w\}$$

nicht regulär ist.

Angenommen L_2 sei regulär.

Dann existiert ein Endlicher Automat $A = (Q, \{0,1\}, \delta, q_0, F)$ mit $L(A) = L_2$.

Wir betrachten die Wörter

$$1, 1^2, ..., 1^{|Q|+1}$$

Per Pigeonhole-Principle existiert $i, j \in \{1, ..., |Q| + 1\}$ mit $i < j$, so dass

$$\hat{\delta}(q_0, 1^i) = \hat{\delta}(q_0, 1^j).$$

Nach Lemma 3.3 gilt nun für alle $z \in \{0,1\}^*$

$$1^i z \in L_2 \iff 1^j z \in L_2$$

Sei $z = 0^j$. Wir haben dann also

$$1^i z = 1^i 0^j \notin L_2,$$

da $i < j$ und ein Wort auch ein Präfix von sich selbst ist (Die Bedingung $|1^i 0^j|_0 \leq |1^i 0^j|_1$ wird verletzt). Aber wir haben auch

$$1^j z = 1^j 0^j \in L_2,$$

was zu einem Widerspruch führt. Also ist die Annahme falsch und L_2 nicht regulär.

■

Einschub - Sprachen mit Einsymbolalphabet

Angenommen es handelt sich bei $L \subseteq \Sigma^*$ um eine Sprache über einem unären Alphabet ($|\Sigma| = 1, \Sigma = \{x\}$).
Dann gilt:

$$\forall w \in \Sigma^* : w = x^{|w|}$$

Insbesondere gibt es für jede Länge nur ein Wort.
Sei die Folge $(w_i)_{i\in\mathbb{N}}$ kanonisch geordnet, so dass $w_i \in L$ (Wenn L endlich betrachten wir nur endlich viele Wörter der Folge).
Durch das gilt folgendes

$$\forall w \in \Sigma^*.\ \forall k \in \mathbb{N}.\ |w_k| < |w| < |w_{k+1}| \implies w \notin L$$

Beispielaufgabe 2 - Pumping Lemma

Zeigen Sie, dass

$$L = \{0^{n\cdot\lceil\sqrt{n}\rceil} \mid n \in \mathbb{N}\}$$

nicht regulär ist.

Angenommen $L = \{0^{0\cdot\lceil\sqrt{0}\rceil}, 0^{1\cdot\lceil\sqrt{1}\rceil}, 0^{2\cdot\lceil\sqrt{2}\rceil}, ...\}$ sei regulär.

Seien $w_0, w_1, w_2, ...$ die Wörter von L in kanonischer Reihenfolge. Nach dem Pumping Lemma gibt es ein $n_0 \in \mathbb{N}$, dass die Bedingungen (i)-(iii) erfüllt sind.

Wir wählen $w = w_{n_0^2} = 0^{n_0^2\lceil\sqrt{n_0^2}\rceil} \in L$.

Es ist leicht zu sehen das $|w| \geq n_0$ und folglich existiert eine Aufteilung $w = yxz$ ($y = 0^l$, $x = 0^m$ und $z = 0^{n_0^2\lceil\sqrt{n_0^2}\rceil - l - m}$), die (i)-(iii) erfüllt.

Da nach (i) $|yx| = l + m \leq n_0$, folgt $|x| = m \leq n_0$.

Aus (ii) folgt $|x| = m \geq 1$.

Wegen $|yx^2z| = |yxz| + |x|$ gilt also $|yxz| < |yx^2z| \leq |yxz| + n_0$.

Das nächste Wort in L nach $w_{n_0^2}$ ist $w_{n_0^2+1}$ und es gilt

$$\begin{aligned}
|w_{n_0^2+1}| - |w_{n_0^2}| &= (n_0^2+1)\cdot\lceil\sqrt{n_0^2+1}\rceil - n_0^2\cdot\lceil\sqrt{n_0^2}\rceil \\
&= (n_0^2+1)\cdot\lceil\sqrt{n_0^2+1}\rceil - n_0^2\cdot n_0 \\
&> (n_0^2+1)\cdot n_0 - n_0^3 \\
&= n_0
\end{aligned}$$

Die strikte Ungleichung gilt da $n_0 \in \mathbb{N}$ und $n_0 = \left\lceil\sqrt{n_0^2}\right\rceil < \sqrt{n_0^2+1} \leq \left\lceil\sqrt{n_0^2+1}\right\rceil$.

$$\implies |w_{n_0^2+1}| \geq |w_{n_0^2}| + (n_0+1)$$

Somit gilt

$$|w_{n_0^2}| < |yx^2z| < |w_{n_0^2+1}|$$

Daraus folgt $yx^2z \notin L$, während $yxz \in L$, in Widerspruch zu (iii).

■

Beispielaufgabe 3 - Kolmogorov Methode

Zeigen Sie, dass

$$L = \{0^{n \cdot \lceil \sqrt{n} \rceil} \mid n \in \mathbb{N}\}$$

nicht regulär ist.

Widerspruchsannahme: Sei L regulär.

Wir betrachten

$$L_{0^{m \cdot \lceil \sqrt{m} \rceil + 1}} = \{y \in \Sigma^* \mid 0^{m \cdot \lceil \sqrt{m} \rceil + 1} y \in L\}$$

Dann ist für jedes $m \in \mathbb{N}$ das Wort

$$y_1 = 0^{(m+1) \cdot \lceil \sqrt{m+1} \rceil - (m \cdot \lceil \sqrt{m} \rceil + 1)}$$

das kanonisch erste Wort der Sprache $L_{0^{m \cdot \lceil \sqrt{m} \rceil + 1}}$.

Nach Satz 3.1 existiert eine Konstante c, so dass gilt

$$K(y_1) \leq \lceil \log_2(1+1) \rceil + c = 1 + c$$

für jedes $m \in \mathbb{N}$.

Da die Länge von $|y_1|$

$$\begin{aligned} |y_1| &= (m+1) \cdot \lceil \sqrt{m+1} \rceil - (m \cdot \lceil \sqrt{m} \rceil + 1) \\ &\geq (m+1) \cdot \lceil \sqrt{m} \rceil - m \cdot \lceil \sqrt{m} \rceil - 1 \\ &= \lceil \sqrt{m} \rceil - 1 \overset{m \to \infty}{\longrightarrow} \infty \end{aligned}$$

beliebig gross werden kann, gibt es unendlich viele Wörter von dieser Form.

Dies ist ein Widerspruch, da es nur endlich viele Programme der Länge maximal $1 + c$ geben kann.

■

6 Nichtdeterministische Endliche Automaten

6.1 Definitionen

Definition NEA

Ein **nichtdeterministischer endlicher Automat (NEA)** ist ein Quintupel $M = (Q, \Sigma, \delta, q_0, F)$. Dabei ist

(i) Q eine endliche Menge, **Zustandsmenge** genannt,

(ii) Σ ein Alphabet, **Eingabealphabet** genannt,

(iii) $q_0 \in Q$ der **Anfangszustand**,

(iv) $F \subseteq Q$ die Menge der **akzeptierenden Zustände** und

(v) δ eine Funktion von $Q \times \Sigma$ nach $\mathcal{P}(Q)$, **Übergangsfunktion genannt**.

Ein NEA kann zu einem Zustand q und einem gelesenen Zeichen a mehrere oder gar keinen Nachfolgezustand haben.

Konfigurationen für NEAs

Eine **Konfiguration** von M ist ein Tupel $(q, w) \in Q \times \Sigma^*$.

- "M befindet sich in einer Konfiguration $(q, w) \in Q \times \Sigma^*$, wenn M im Zustand q ist und noch das Suffix w eines Eingabewortes lesen soll."

- Die Konfiguration $(q_0, x) \in \{q_0\} \times \Sigma^*$ ist die **Startkonfiguration für das Wort** x.

Ein **Schritt** von M ist eine Relation (auf Konfigurationen) $\vdash_M \subseteq (Q \times \Sigma^*) \times (Q \times \Sigma^*)$, definiert durch

$$(q, w) \vdash_M (p, x) \iff w = ax, a \in \Sigma \text{ und } p \in \delta(q, a)$$

Berechnungen für NEAs

Eine **Berechnung von M** ist eine endliche Folge $C_1, ..., C_k$ von Konfigurationen, so dass

$$C_i \vdash_M C_{i+1} \text{ für alle } 1 \leq i \leq k.$$

Eine **Berechnung von M auf x** ist eine Berechnung $C = C_0, ..., C_m$, wobei $C_0 = (q_0, x)$ und **entweder** $C_m \in Q \times \{\lambda\}$ **oder** $C_m = (q, ay)$ für ein $a \in \Sigma, y \in \Sigma^*$ und $q \in Q$, so dass $\delta(q, a) = \emptyset$.

Falls $C_m \in F \times \{\lambda\}$, sagen wir, dass C eine **akzeptierende Berechnung** von M auf x ist, und dass M **das Wort** x **akzeptiert**.

Die Relation $\vdash_M^*$ ist die reflexive und transitive Hülle von $\vdash_M$, genau wie bei einem EA.

Wir definieren

$$\mathbf{L(M)} = \{w \in \Sigma^* \mid (q_0, w) \vdash_M^* (p, \lambda) \text{ für ein } p \in F\}$$

als die **von M akzeptierte Sprache.**

Zu der Übergangsfunktion δ definieren wir die Funktion $\hat{\delta} : (Q \times \Sigma^*) \to \mathcal{P}(Q)$ wie folgt:

(i) $\hat{\delta}(q, \lambda) = \{q\}$ für alle $q \in Q$

(ii) $\hat{\delta}(q, wa) = \bigcup_{r \in \hat{\delta}(q,w)} \delta(r, a)$ für alle $q \in Q, a \in \Sigma, w \in \Sigma^*$.